Debreceni Egyetem

Informatikai Kar

SUDOKU ÉS HEURISZTIKA

Témavezető:

Dr. Aszalós László

Egyetemi adjunktus

Készítette:

Bónizs Attila

Programtervező Matematikus

Debrecen

2009

Tartalomjegyzék

1. Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, Dr. Aszalós László egyetemi adjunktusnak, hiszen odaadó segítsége, végtelen türelme és értékes tanácsai nélkül diplomamunkám nem készülhetett volna el. Köszönettel tartozom családomnak, barátaimnak, akik mindvégig türelmesek és segítőkészek voltak velem szemben.

Végül, de nem utolsó sorban szeretnék köszönetet mondani egyetemi oktatóimnak, akik nagymértékben járultak hozzá elméleti tudásom elmélyítéséhez.

2. Bevezetés

Diplomamunkám célja a Sudoku nevű logikai játék megoldását célzó heurisztikák, stratégiák, algoritmusok ismertetése, és számos stratégia implementálásával egy megoldó program létrehozása. A program létrehozásánál a fő szempont, hogy e népszerű játék rejtvényeinek megoldását tisztán a Sudokut ismerő és sokat játszott emberek megfigyeléseiből, tapasztalataiból származó, heurisztikus algoritmusok segítségével képes legyen előállítani. Ezek a heurisztikák, vagy, ahogy az irodalmakban sok helyen említik, stratégiák [12], felruházzák a programot némi Mesterséges Intelligenciával. A bemutatásra kerülő stratégiák, elemzéseken, megfigyeléseken alapulnak, mellőzik a számítógép erejére támaszkodó próbálgatást.

Jelenleg is rengeteg embert foglalkoztat, hogy milyen módszerek alkalmazásával lehet megoldani egyes feladványokat. Egy Sudoku feladatot számítógéppel egyszerűen meg lehet oldatni. Egyik legegyszerűbb módja ún. „brute-force” algoritmus használata [4]. Az ilyen algoritmus lépked végig a cellákon, az üres cellába beírja az első lehetőséget, az 1-est, ellenőrzi, hogy ütközik-e valami szabályba, ha nem, lép a következő cellára oda is beírja az 1-est és ellenőriz, ha szabályba ütközik, akkor írja a következő számot és így tovább. Ha az utolsó beírt számnál is ütközés van, akkor törli és visszalép az előző cellára, egyel nagyobb értéket ad neki. Ezt addig folyatja, míg ki nem tölti a teljes feladatot. Egy ilyen algoritmus forráskódját röviden meg lehet írni (például három soros Perl algoritmus: 1. ábra [6]), viszont a visszalépések miatt rengeteg munkát végezhet. Egy heurisztikus algoritmus sokkal izgalmasabb és érdekesebb, valamint alkalmazásukkal csökkenthetjük a lépések számát.

```
use integer;@A=split//,<>;sub R{for$i(0..80){next if$A[$i];my%t=map{$_/9
==$i/9||$_%9==$i%9||$_/27==$i/27&&$_%9/3==$i%9/3?$A[$_]:0=>1}0..80;R($A[
$i]=$_)for grep{!$t{$_}}1..9;return$A[$i]=0}die@A}R
```

1. ábra: Perl algoritmus.

3. A Sudoku

3.1. Története

A rejtvény ma ismert változatát Howard Garns nyugdíjas építész és szabadúszó rejtvénytervező alkotta meg. Először 1979-ben publikálta a Dell Magazine, akkor még „Number Place” néven. Sokak szerint az ötletet Leonhard Euler „latin négyzet”-tel való munkássága ihlette. A latin négyzet egy n * n-es táblázat, amelynek soraiban és oszlopaiban n különböző szimbólum szerepel oly módon, hogy ezek mindegyike minden sorban és oszlopban pontosan egyszer fordul elő.

1984-ben a Nikoli kiadó bemutatta a játékot Japánban, a következő névvel: 数字は独身に限る, *Suuji wa dokushin ni kagiru* (jelentése: „a számjegyek csak egyszer szerepelhetnek”). Valamivel később Maki Kaji rövidítette a nevét Sudokura. A Föld többi részén 2005 környékén lett közkedvelt. A mai világban már szinte mindenki hallott róla és rengetegen teszik próbára tudásukat egy-egy könnyebb vagy nehezebb feladvánnyal. Számos újságban jelennek meg hetente, vagy naponta feladványok és több, csak Sudoku feladványokat tartalmazó könyv is jelent már meg. Az Interneten is számtalan weboldal van, ami csak a Sudokuval és megoldásával foglalkozik.

3.2. Játékszabályok

Az általános, és legelterjedtebb Sudoku egy 9 * 9-es tábla, ami kilenc darab 3 * 3-as blokkból áll. A tábla celláiban úgy kell elhelyezni a számokat 1-től 9-ig, hogy azok minden sorban, oszlopban és blokkban pontosan egyszer szerepelnek. Kezdéshez néhány cella értéke adott (példa: 2. ábra). A rejtvény lényege, hogy egy hiányosan kitöltött táblát kitöltsünk a szabályoknak megfelelően. A szabályos Sudoku feladatoknak egyetlen megoldása van, és ellentmondásmentesen van megadva.

Az évek alatt a kezdeti szabályok változtatásával, vagy új szabályok hozzáadásával újfajta rejtvényeket állítottak elő. Ilyenek például a kisebb és nagyobb táblás verziók. Ezek is n * n-es táblák, és 1-től n-ig tartalmaznak számokat. A blokkokban lévő cellák száma ugyanúgy n mint az alap verziónál, de a méret miatt ez lehet, hogy nem négyzet alakú. Példák a kisebb és

nagyobb Sudokukra: 4 * 4-es tábla (2 * 2-es blokkokkal), 12 * 12-es tábla (3 * 4-es blokkal), 25 * 25-ös tábla (5 * 5-ös blokkokkal).

4	6				1			
		2		9	6			
	3						6	8
							3	7
			6		7			
5	1							
8	4						5	
			7	1		9		
			3				2	4

4	6	5	8	3	1	2	7	9
7	8	2	4	9	6	3	1	5
1	3	9	5	7	2	4	6	8
6	9	4	1	2	5	8	3	7
3	2	8	6	4	7	5	9	1
5	1	7	9	8	3	6	4	2
8	4	1	2	6	9	7	5	3
2	5	3	7	1	4	9	8	6
9	7	6	3	5	8	1	2	4

2. ábra: Sudoku feladat és megoldása.

A Sudoku-X-ben [9] annyival egészül ki az alap feladat, hogy az átlókban is ugyanolyan szabály szerint kell kitölteni, mint a sorokat, oszlopokat, blokkokat.

Killer Sudoku [9] rejtvény megadás különösen érdekes. A kezdő táblán általában kevés szám szerepel, de kapunk egy második táblát is. Ezen különböző színekkel jelölt halmazok vannak, melyekben szerepel egy érték, ami azt mondja meg, hogy az adott halmazban mennyi az ott előforduló értékek összege.

A Jiggsaw Sudoku [9] esetében az a különbség a blokkok alakjában van. Általában nem négyszög alakot vesznek fel.

3D Sudoku [10] lényegében egy Rubik kocka, ahol színek helyett számok szerepelnek 1-től 9-ig, és az a cél, hogy minden oldalon pontosan egyszer szerepeljen minden szám.

4. Heurisztikák ismertetése

4.1. Heurisztika

A heurisztika görög heureszisz (rátalálás) szóból származik. Egy megoldási módszer, ami az egyértelmű algoritmusok helyett korábban megszerzett tapasztalatok felhasználásával működik. A matematikában heurisztikusnak az olyan gondolatmenetet nevezik, amely még nem bizonyított állítások mellett érvel tapasztalati tényekből megfogalmazott tulajdonságok segítségével. Ha heurisztika útján sikerül egy problémát megoldani, akkor a bizonyítás már szigorú logikai úton is elvégezhet.

Sudoku rejtvény heurisztikus megoldásán azt értem, hogy sok ember korábban megszerzett tapasztalatának, tudásának felhasználásával létrejött módszerek segítségével lépésről lépésre könnyítsük a rejtvényt, míg végül sikerül megoldani. Ezeket a módszereket hívom heurisztikáknak, heurisztikus algoritmusoknak. Angol nyelvű irodalmakban stratégiákként (strategy) említik őket [12].

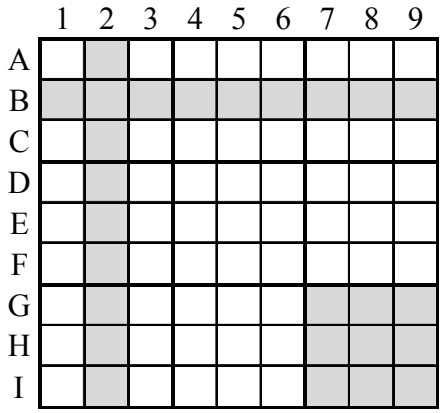
Egy Sudoku feladat megoldása a következőt jelenti: az indulásnál rendelkezésünkre álló cellákat, és az ismert szabályokat felhasználva kitöltsük a többi cellát azzal az egy értékkel, ami ott szerepelhet. Egy konkrét érték meghatározása komplex feladat, ezért egy-két kivétellel az összes stratégia arra irányul, hogy az egy cellában előforduló jelöltek számát csökkentsük. Kevesebb jelölttel nagyobb esélyünk van egy mező értékének meghatározására.

4.2. Elnevezések

A tárgyalandó heurisztikus stratégiák neveit minden esetben angol nevével hivatkozzák, ezért a diplomamunkában én is ezeket fogom használni. Mivel ezeket a módszereket nem egyetlen ember találta ki, ezért egyes stratégiákat esetleg más néven is ismerhetünk.

A feladat egy tábla (board), ami nyolcvanegy darab cellát, mezőt (cell) tartalmaz. A szabályok szerint speciális kapcsolat áll fenn az olyan cellák között, amelyek egy sorban (row), oszlopban (column), vagy egy blokkban (box) helyezkednek el, az ilyen mezőkre úgy hivatkozunk, hogy egy házban (unit) vannak (3. ábra) [7]. A cellának az értéke lehet ismert (amennyiben megadtuk kiindulásnál, vagy már meghatároztuk), vagy ismeretlen. Egy ismeretlen

cella értékének jelöltjei (candidate) vannak. A cellában szereplő jelöltek tehát lehetséges értékek, amiket később a cella felvehet értékéül. A kiindulási tábla tartalmaz előre megadott értékű cellákat, a többi - üres - cella tartalmazza az összes lehetséges értéket 1 és 9 között.



3. ábra: Sor, oszlop, és blokk; három különböző típusú példa házra.

Érdemes megfigyelni, hogy az alaplépést leszámítva az összes heurisztika csak az üres cellákban található jelöltek elhelyezkedését figyeli, és abból próbál következtetéseket levonni.

4.3. Alaplépés

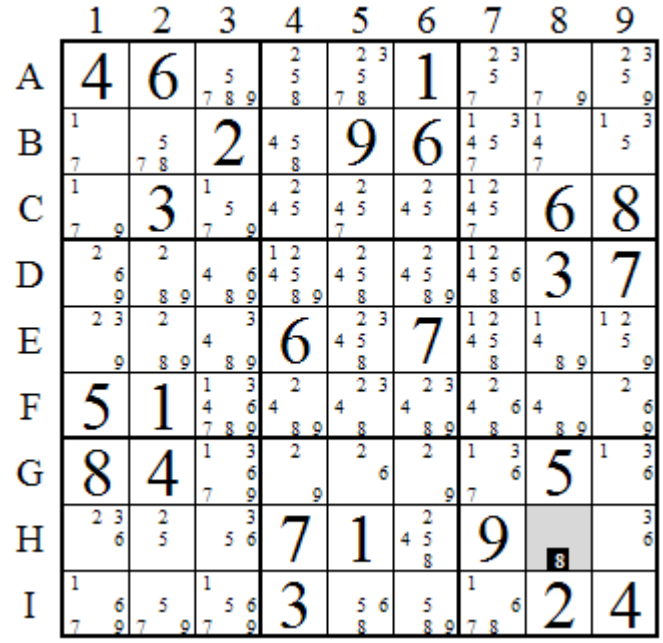
Amennyiben egy cella értéke ismert, nincsenek jelöltjei, a vele egy sorban, oszlopban és blokkban lévő üres cellák jelöltjei közül eltávolíthatjuk ezt az értéket. Ezt a lépést meg kell ismételni minden olyan alkalommal, amikor egy cella értéket kap. A programban a következőképpen zajlik a lépés: Sorfolytonosan bejárjuk a táblát, ha olyan cellához érünk, aminek van értéke, akkor azt az értéket a cellával egy sorba, oszlopba, vagy blokkba, azaz egy házba tartozó mezők jelöltjei közül eltávolítjuk.

4.4. Naked Single

A stratégia nevében a Naked (magyar előfordulásban „csupasz”, „tiszta”) kifejezés arra utal, hogy az üres cella minden jelöltjét, kandidáltját figyelembe vesszük.

Olyan mezőket keresünk, amelyekben csak egyetlen jelölt van, ekkor ezt az értéket veszi fel a mező (4. ábra). Az alaplépéssel együtt gyakorlatilag ez a legfontosabb lépés, egyszerűsége

miatt az alaplépéssel együtt került implementálásra. A tábla bejárása alatt, ha egy cellában egyetlen jelölt van, akkor beállítjuk azt a cella értékének és az alaplépés is lefut.



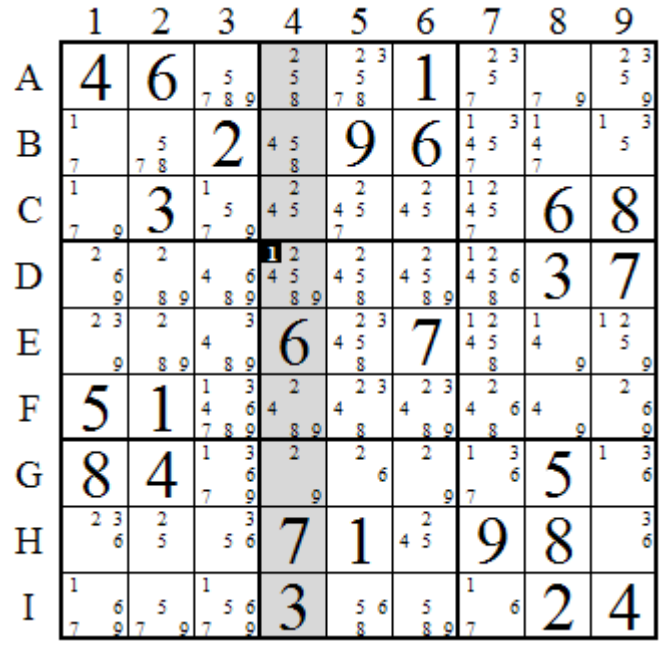
4. ábra: H8 cellában 8 egyedüli jelölt, Naked Single. A cella felveszi az értéket.

4.5. Hidden Single

Megvizsgáljuk, hogy házon (sor, oszlop, blokk) belül van-e olyan jelölt, ami pontosan egy cella kandidáltjai között szerepel. Ha van, a cella felveszi ezt az értéket (5. ábra).

Megvalósítás: Bejárjuk külön az összes sort, oszlopot, blokkot és az összes jelöltre ellenőrizzük, hogy hányszor szerepel az adott házban, ha csak egyszer a cellája felveszi az értékét.

A további heurisztikákon alapuló algoritmusok nem a cellák azonnali kitöltésére szolgálnak, hanem az üres cellákban található jelöltek számának csökkentésére. A cél a jelöltek számát addig csökkenteni, amíg a megoldó szoftver az előző algoritmusokat használva ki nem tölti a teljes feladatot.



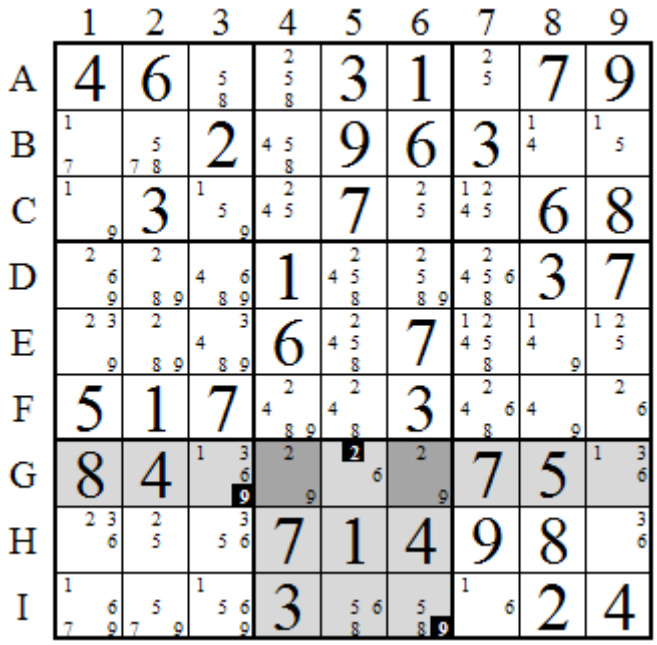
5. ábra: A 4-ik oszlopban az 1-es érték csak D4-ben szerepel, Hidden Single. A cella felveszi az értéket.

4.6. Naked Pair

A stratégia alapja, hogy olyan cella párokat keresünk, amelyek egy házban találhatók, két lehetséges értéket tartalmaznak és ugyanazt a kettőt. Amennyiben létezik ilyen pár a két érték mindenképp azokban a mezőkben lesznek, így ezek az értékek a házon beül eltávolíthatók a többi cella jelöltjei közül (6. ábra). A programban a metódus sorfolytonosan járja be a táblát, ha az aktuális cella két jelöltet tartalmaz, keresünk vele egy házban lévő ugyanazt a két jelöltet tartalmazó mezőt, ha van, az algoritmus eltávolítja a házban lévő többi üres cella jelöltjei közül ezt a két számot.

A Naked Pair stratégiának létezik Naked Triple, és Naked Quad kiterjesztett változata is. Ezek a heurisztikák már bonyolultabbak. A Naked Triple nevéből is láthatóan két cella és két lehetséges érték helyett három cellával, és ezekben három különböző jelölttel foglalkozik. Ezek a cellák ugyanúgy azonos házon belül helyezkednek el. A bonyolultságát az adja, hogy nem kell egyszerre mind a három jelöltnek mind a három cellában szerepelnie. A mezőkben szerepelhet két vagy három jelölt is, ha továbbra is fennáll az, hogy a három cellában csak a három érték kombinációja szerepelhet. Előállhatnak például az alábbi kombinációk: (123)(123)(123), (123)(123)(12), (123)(12)(23), stb., és a legérdekesebb forma (12)(23)(13), későbbiekben látni fogjuk majd, hogy ezt a szerkezetet használja az Y-Wing stratégia is. A Naked Quad hasonlóan

épül fel, mint a Naked Triple, kiegészülve még egy cellával és jelölttel. Előfordulása meglehetősen ritka.



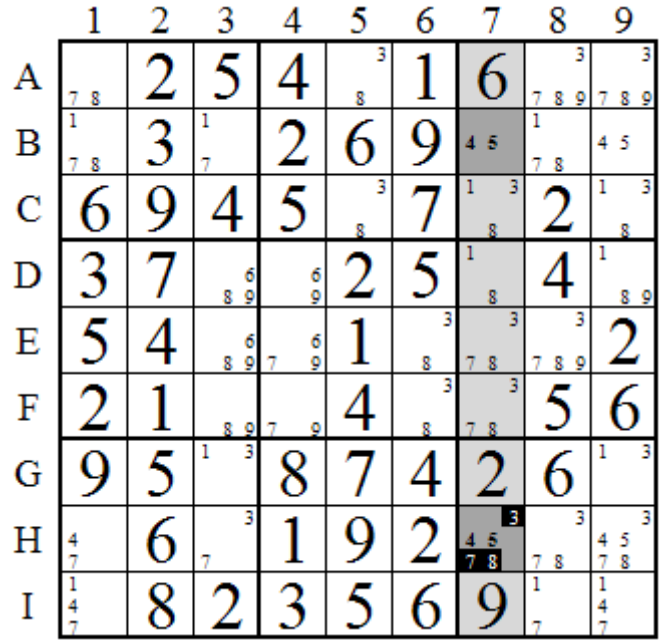
6. ábra: G4 és G6 Naked Pair egy sorban és blokkban is; eltávolítható G3-ból 9, G5-ből 2 és I6-ból 9.

4.7. Hidden Pair

A Hidden („rejtett”) szó arra utal, hogy egy cella vizsgálatánál, eliminálási lehetőség keresésénél nem használjuk az összes cellában található jelöltet, csak néhány kitüntetett értékkel foglalkozunk. A Hidden Pair stratégia a Naked Pair-hez hasonlóan két jelöltre támaszkodik. A cél olyan két cellát találni, ami tartalmazza ezt a két jelöltet, egy házban vannak és ezen a házon belül egyetlen más cella sem tartalmazza a két jelölt bármelyikét. Találat esetén nem tudjuk még megmondani, hogy melyik mező melyik értéket fogja tartalmazni, de a két cellának mindenképpen tartalmaznia kell a két jelölt egyikét, ezért azokból a cellákból eltávolíthatjuk a többi lehetséges értéket (7. ábra).

Az implementált algoritmus működése: Sorfolytonosan bejárjuk a Sudoku táblát. Ha az aktuális mező nincs kitöltve, végig lépkedünk a lehetséges értékein és olyan mezőt keresünk vele egy házban (a folyamatot külön elvégezzük sorra, oszlopra, blokkra), amivel az aktuális lehetséges értéket csak ketten tartalmazzák az adott házban. Miután megtaláltuk az ilyen cellákat, már egyszerű feladat megvizsgálni, hogy ezek között van-e olyan, ami tartalmaz két olyan lehetséges értéket is, mint az aktuálisan vizsgált mező, ha van ilyen, előállt egy Hidden

Pair. Ekkor eltávolítjuk a két mezőből a két lehetséges értéken kívül az összes többi értéket, és ha megfigyeljük ezzel a lépéssel előállt egy Naked Pair, így valószínű, hogy a további eliminálásokat hajthatunk végre. A Hidden Pair alkalmazása és ezzel Naked Pair-ek feltárása nagyon fontos a feladat megoldása szempontjából, mert előkészíti a táblát más heurisztikák használatához.

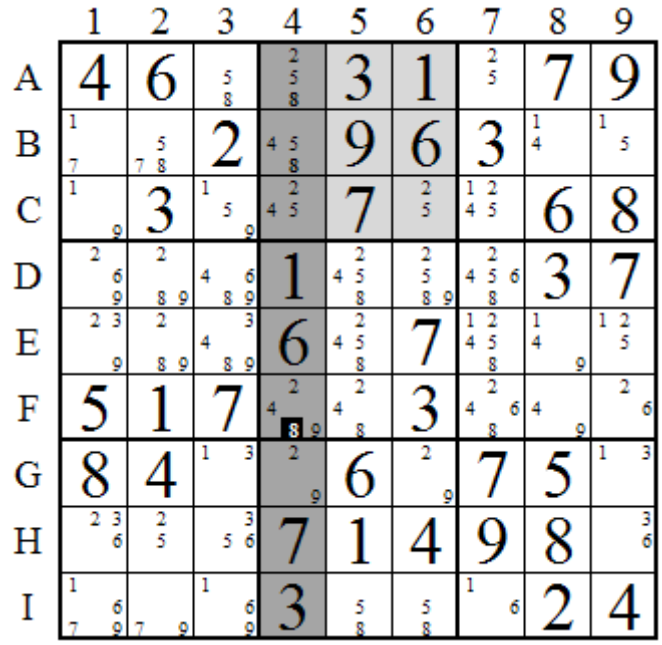


7. ábra: B7 és H7 Hidden Pair a 7-ik oszlopban; 3, 7, 8 eltávolítható H7-ből.

A Hidden Pair stratégia is kibővíthető úgy, hogy ha több értéket veszünk figyelembe, kiterjesztése ugyanúgy működik, mint a Naked Pair-nél. Hidden Triple-nél és Hidden Quad-nál három és négy cellában három és négy jelölt kombinációját keressük.

4.8. Pointing Pair

Az összes blokkokat külön vizsgáljuk; ha egy lehetséges érték a blokkon belül többször is előfordul, de az előfordulások egy sorra, vagy egy oszlopra esnek, akkor annak az értéknek abban a sorban, vagy oszlopban mindenképp ezen a blokkon belül kell lennie, ezért a sor vagy oszlop blokkon kívül eső részéből eltávolíthatjuk a jelöltet (8. ábra).



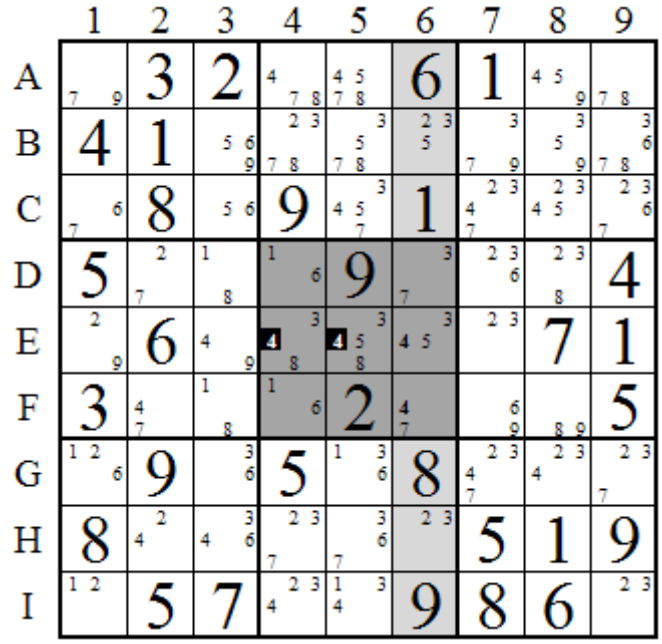
8. ábra: A második blokkban a 8-as jelölt csak a 4-ik oszlop mentén található meg (A4, B4), a 4-ik oszlopból a blokkon kívül eltávolítjuk a 8-as jelölt többi előfordulását (F4).

A függvény működése a programban: blokkonként vesszünk sorba a lehetséges értékeket, ha a jelölt előfordul a blokkon belül, a tartalmazó cella koordinátáit vizsgálva megállapítja, hogy egy sorra vagy oszlopra esnek-e. Ha a vizsgálat igaz a jelöltet eltávolítjuk a megfelelő helyekről.

4.9. Box Line Reduction

Vizsgáljuk külön az összes sort és oszlopot; ha egy jelölt egy soron vagy oszlopon belül több cellában is előfordul, de ezek egy blokkon belül vannak, akkor a lehetséges érték azon a blokkon belül csak erre a sorra, vagy oszlopra eshet, a blokk ezen kívül eső részéből törölhetjük a jelölt többi előfordulását (9. ábra).

A stratégia megvalósítása hasonló az előzőhöz, blokk - sor, blokk – oszlop kapcsolat helyett oszlop – blokk, sor – blokk kapcsolatokban végezzük el a vizsgálatokat és a törlést, ha lehetséges.



9. ábra: A 6-odik oszlopban a 4-es jelölt csak az ötödik blokkban fordul elő (E6, F6): eltávolítjuk az ötös blokkból azokat a 4-es jelölteket, amik a 6-os soron kívül találhatók (E4, E5).

Az előző két heurisztikát együtt Intersection Removal néven is szokták említeni, mert mindkét esetben két halmaz metszetét vizsgáljuk.

4.10. X-Wing

Veszünk egy jelöltet, ami két különböző sorban egyaránt pontosan két cellában fordul elő és ezek a cellák ugyanarra a két oszlopra is esnek. Ebben az esetben előáll egy téglalap, mind a négy sarkában megtalálható a jelölt érték. Ekkor a cellák átlós alakban fogják felvenni az értéket, így az oszlopok a két soron kívül eső mezőiből eltávolíthatjuk a jelöltet. A két sort oszlopra, az oszlopokat sorra felcserélve is alkalmazható ez a stratégia (10. ábra).

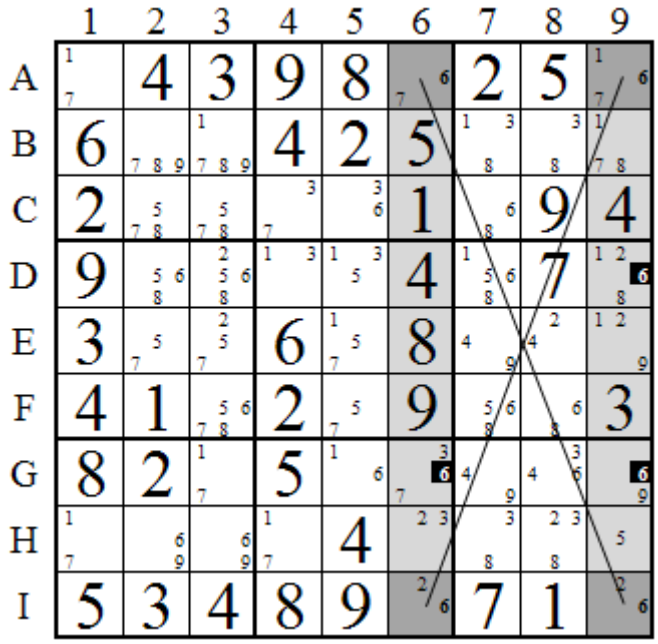
Az X-Wing stratégia általánosítható, így nem csak sorokra és oszlopokra, de blokkokkal is használható. Általánosítva a szabály a következőképp néz ki:
Ha teljesül, hogy

- pontosan két cella van, ami a jelöltet tartalmazza mindkét, ugyanolyan típusú házban (sor, oszlop blokk),
- és ezek a jelöltet tartalmazó cellák két ugyanolyan típusú házban, de az előző típustól eltérőben is benne vannak,

akkor az utóbbi két házból eliminálható a jelölt többi előfordulása. Ezzel az általánosítással hat kombináció jön létre:

- kiindulunk két sorból, és két oszlopból eliminájuk a jelöltet – Classic X-Wing,
- kiindulunk két oszlopból, és két sorból eliminájuk a jelöltet – Classic X-Wing,
- két blokkból indulunk ki, és két sorból eliminálunk,
- két blokkból indulunk ki, és két oszlopból eliminálunk,
- két sorból indulunk ki, és két blokkból eliminálunk,
- két oszlopból indulunk ki és két blokkból eliminálunk.

A harmadik és negyedik sor hatása megegyezik a Box Line Reduction stratégia hatásával, az utolsó két sor hatása pedig megegyezik a Pointing Pairs stratégia hatásával.

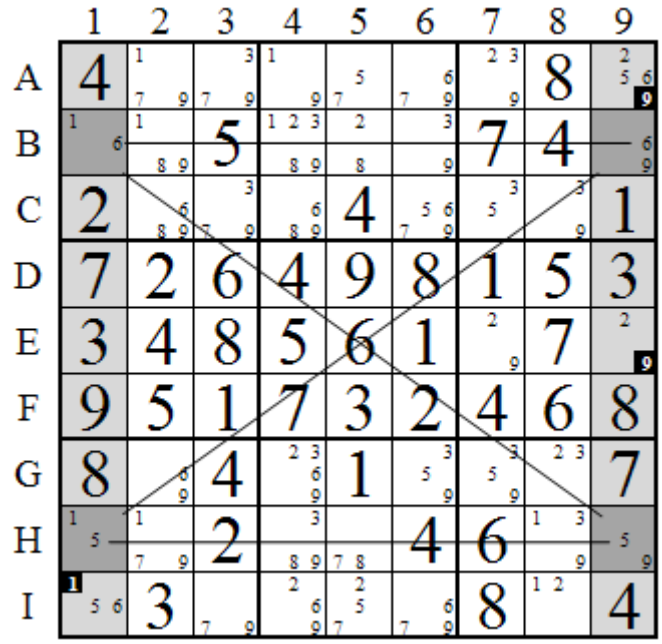


10. ábra: A6, A9, I6, I9 X-Wing 6-ra: a jelöltet töröljük D9, G6, G9 mezőkből.

Az algoritmus működése: Vesszük a jelölteket sorba. Megyünk végig a sorokon és oszlopokon, ha az éppen vizsgált sorban, vagy oszlopban csak kétszer szerepel a jelölt, akkor elraktározzuk a mező párokat külön, annak megfelelően, hogy egy sorban vagy egy oszlopban vannak-e. Az elraktározott mező párok között olyanokat keresünk melyek X-Wing-nek megfelelő téglalapot hoznak létre. A keresést sorból és oszlopból kiindulva is megtörténik. Találat esetén megtörténik az eliminálás.

Az X-Wing stratégiának létezik egy speciális változata, ami abban különbözik az eredetitől, hogy a két házban, amelyet első sorban vizsgálunk nem ugyan az a két jelölt szerepel pontosan kétszer. Több jelöltet vizsgálunk, innen ered az elnevezése is: Multivalue X-Wing. A stratégia működését a 11. ábrán látható példával szemléltetem. Előáll az alábbi X-Wing négyszög, de az második sorban 6-os pár a nyolcadik sorban 5-ös pár áll a középpontban. Ami mégis összeköti a cellákat az oszlopokban az, hogy a 6-os és 5-ös jelöltek mellett ugyanaz a jelölt található, az első oszlopban 1, a kilencedik oszlopban 9. Figyeljük meg, hogy az oszlopokban további 1-es és 9-es található, ezeket tudjuk majd eltávolítani.

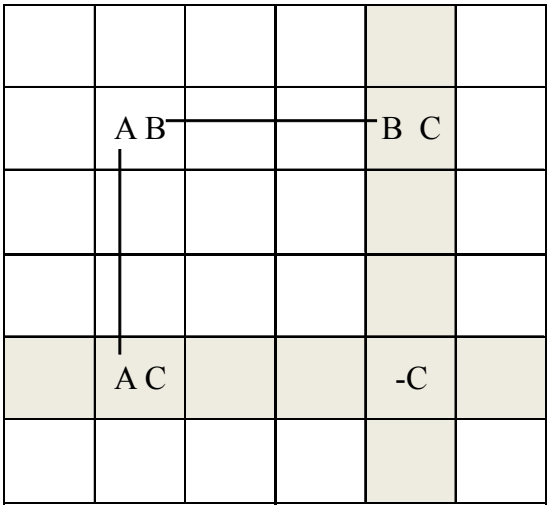
A logika a következő: a második sorban a 6-os értéket mindenképp felveszi a két jelölt cella egyike, ugyanez a helyzet a nyolcadik sorban az 5-ös értékkel. Ez egyértelmű, de a 6-os és 5-ös nem szerepelhet ugyanabban az oszlopban. Bizonyítás: tegyük fel, hogy mindketten az első oszlopban szerepelnek, ebben az esetben, az utolsó oszlopban, két cellában csak a 9-es érték szerepelhetne, ami nem lehetséges, ugyan ez a helyzet, ha az utolsó oszlopban szerepelnek. Emiatt a két értéket csak az átlósan elhelyezkedő két cella veheti fel, ez azt eredményezi, hogy a másik két sarokban 1-es és 9-es szerepel, eliminálva ezzel az adott oszlopban a többi előfordulásukat.



11. ábra: Multivalue X-Wing.

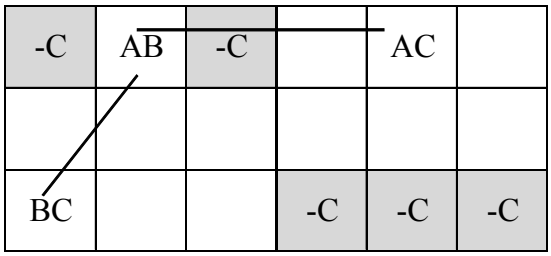
4.11. Y-Wing

Meglehetősen jó jelölt elimináló. A neve arra utal, hogy hasonlít az X-Wing stratégiára, de ebben az esetben három cellát figyelünk a négy helyett, a negyedik sarokból (később látni fogjuk, hogy ez több cella is lehet) távolíthatunk el jelöltet.



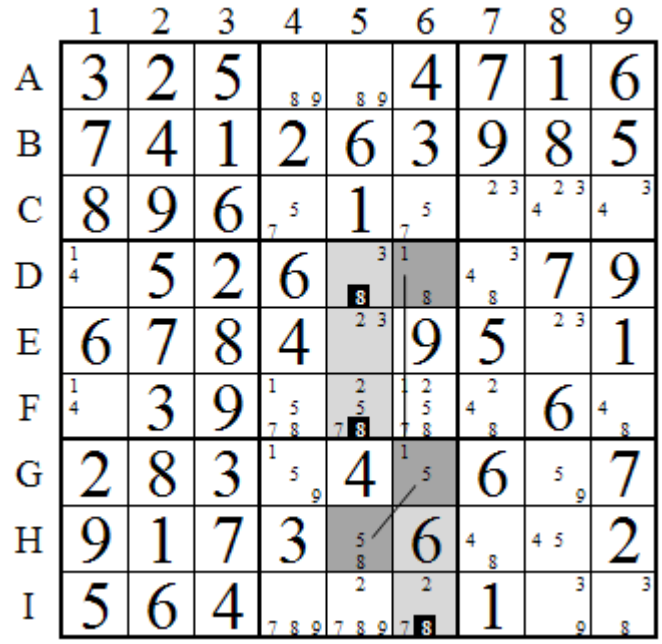
12. ábra: Y-Wing ABC jelöltekkel.

A három cellában háromféle különböző jelölt szerepel a következőképpen: Legyen A, B és C a három különböző lehetséges érték. A háromszögben szereplő összes cellában pontosan két-két jelölt szerepel. A kulcsszerepet játszó cellában legyen AB. A háromszög másik két cellája AC-t és BC-t tartalmaz, és az AB-t tartalmazó cellával külön-külön egy házba van, de egymással nem (12. ábra). Ha az AB cella értéke A lesz, akkor az AC cella C értékű lesz, ha AB B értéket veszi fel, akkor BC cella veszi fel a C értéket. Akármelyik értéket is veszi fel AB, a C értéket mindenképp felveszi egy a másik két mezőből. Ennek az eredménye, hogy az AC, és BC mező által egyszerre látott mezőkben nem szerepelhet a C jelölt, ezért onnan eltávolíthatjuk.



13. ábra: Y-Wing.

Az Y-Wing 12. ábrán látható esetben csak egyetlen cellát lát együtt AC és BC, ezért csak onnan távolíthatjuk el a jelöltet. Ennél érdekesebb eset, amikor a három cella „közelebb” helyezkedik egymáshoz, azaz AB az egyik cellával egy blokkban van (13. ábra). Ebben az esetben BC látja az AB kulcscellát, mert egy blokkban vannak, AC pedig egy sorban van AB-vel, ezért az Y-Wing teljesül. A BC és AC cellák által közösen látott cellák száma pedig AB cellával együtt hat. A -C-vel jelölt cellákból eliminálhatjuk a C értékű jelöltet.



14. ábra: G6, D6, H5 Y-Wing, 1 és 5 jelöltekkel a központban; töröljük D5, F5, I6 mezőkből 8-at.

Y-Wing implementálása nagyvonalakban: Bejárjuk a táblát, csak olyan cellával foglalkozunk, amelyekben pontosan két lehetséges érték szerepel. Ha találunk ilyet, ezzel a cellával egy sorban, oszlopban (a sor és oszlop csak azon részét vizsgáljuk, amelyik kívül esik az eredeti cellát tartalmazó blokkon) és blokkban (a blokk azon részét nézzük, amelyik nincs egy sorban és oszlopban az eredeti cellával), házanként külön eltároljuk az olyan cellákat, amelyekben szintén pontosan két jelölt szerepel, de úgy, hogy a kiindulási cellával pontosan egy ugyanolyan jelöltjük van. Ezután az eltárolt cellák között keresünk olyan cella párokat, amelyek külön házban vannak és két jelöltjük közül, amelyiket nem tartalmazza az eredeti cella, az különbözik, a másik jelöltjük viszont ugyanaz. Ha találtunk ilyen cellapárt, akkor az eredetivel a középpontban előállt egy Y-Wing. Ezután a feladat eltávolítani a két cella házainak metszetéből azt a bizonyos harmadik értéket.

Az Y-Wing stratégiának létezik egzotikusabb változat is. Ebben az esetben az Y-Wing-et lánccá bővítjük, innen ered a neve is: Y-Wing Chain. Nem egyetlen központi cella kapcsolja össze a két szélső cellát, a helyét átveszi egy lánc, amelyben a szomszéd párok Naked Pair-t alkotnak, azaz mindegyik AB. Az alap Y-Wing-et tekinthetjük egy hosszúságú láncú Y-Wing Chain-nek.

Az eddig tárgyalt stratégiák nagyon erős stratégiák. Nem titkolt cél, hogy a program ezen stratégiák felhasználásával képes legyen a tömegeknek szánt Sudoku rejtvények döntő többségét megoldani.

Az eddig említett heurisztikákon kívül létezik még számos másik. Ezek a heurisztikák bonyolult felépítésűek, sokszor több összefüggést vizsgálnak, mint amennyit egy átlagember egyszerre fejben tud tartani. Felhasználni többnyire csak rendkívül nehéz feladatok megoldására lehet. Akadnak közöttük olyanok, melyeknek eredményei hasonlók, fedésben vannak egymással, vagy akár valamelyik tárgyalt módszerrel. Az ilyenek sok esetben nem használhatók egyszerre, mert elvehetik egymástól a lehetőséget, eredményeikkel eltűntethetik a mintát, amit a másik módszer keres.

Létezik még egy speciális csoport, amit „Trial and Error” gyűjtő néven említenek [11]. Olyan stratégiák tartoznak ebbe a csoportba, amelyek gyakorlatilag tippeléssel próbálják kikövetkeztetni egy cella értékét. Ez a csoport megosztja az embereket, egyik oldal szerint a találgatás mivolta miatt ezek a stratégiák nem igazi megoldó módszerek, mert nem egy megfigyelt, meghatározott minta alapján érünk vele eredményt. A másik oldal szerint teljesen megfelel a követelményeknek, mivel emberi gondolkodásmód alapján jöttek létre ezek a stratégiák is. Az általános Sudoku feladványok megoldhatóak logikus lépésekkel nem szükséges találgatni. Az emberek azonban találgatással is próbálkozhatnak, ha elakadtak a megoldás folyamán. Ilyen stratégia például a Nishio. Működése: Olyan cellát keresünk, amiben kevés jelölt van. Kiválasztjuk az egyiket értékének, és így próbáljuk megoldani a rejtvényt. Amennyiben sikerül megoldani a rejtvényt, szerencsénk volt. Ha ellentmondásba ütközünk (pl.: egy üres cellában nincs jelölt, amit felvehetne értékül, vagy egy házban kétszer szerepelne egyetlen érték), tudjuk, hogy azt az értéket nem veheti fel a cella. Ha nem ütközünk ellentmondásba, de a rejtvényt sem sikerült így megoldani, akkor nem tudtunk meg semmit, nincs törölhető jelölt. A stratégiánál sok múlik azon, hogy melyik lehetséges értéket választjuk, ha hamar ellentmondásra jutunk, kihúzhatjuk az értéket a cellából, ha a jó értéket választottuk, az csak akkor derül ki, ha

megoldottuk a teljes rejtvényt. A módszer úgy működik, mint egy backtrack algoritmus, azonban a célunk hogy ellentmondást találjunk. A legtöbb implementáció csak néhány lépést tesz a választás után, ha nem jut ellentmondásra próbálkozik másik jelölttel.

5. Sudoku rejtvényt megoldó program

5.1. A program célja

Munkám során célom egy olyan program létrehozása volt, amely heurisztikák használatával képes Sudoku feladatok megoldására. A programmal szemben támasztott fő követelmény, hogy alkalmas legyen nagy mennyiségű rejtvény megoldásán túl az alkalmazott heurisztikákról statisztikák készítésére. Mivel nem egy játékprogram létrehozása volt a célom, a megjelenítésre kevesebb figyelmet fordítottam.

5.2. Szoftver követelmények

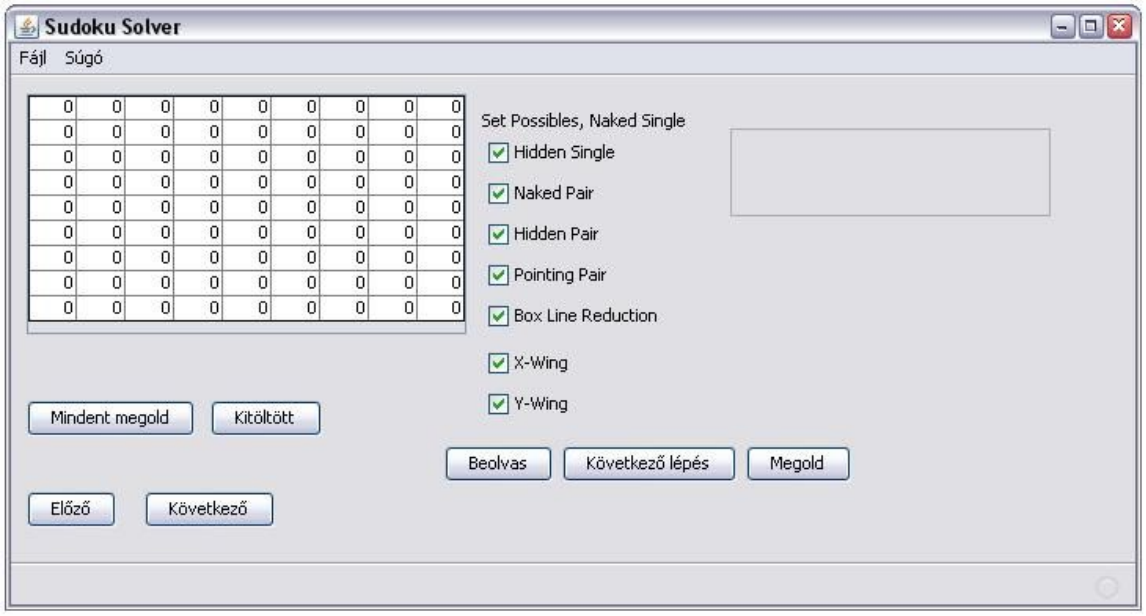
A programot Windows XP operációs rendszeren, NetBeans IDE 6.5 környezetben fejlesztettem Java SE Development Kit (JDK) 6-os verziója mellett. A program teljes NetBeans projektje a forráskóddal és futtatható a builddel együtt megtalálható a mellékelt cd-n. A program futtatásához a futtató gépen szükség van Java Runtime Environment (JRE) 6-ra. Korábbi verziókkal probléma léphet fel, mert a forráskód olyan elemeket is tartalmaz, amelyek nem részei a korábbi verzióknak.

A programot (SudokSolver mappa és teljes tartalma) az indítás előtt másoljuk fel a cd-ről merevlemezre, így a kimeneti fájl írásakor nem lesz probléma. A programot SudokuSolver.jar fájllal indíthatjuk, ami a \SudokuSolver\dist\ mappában található. Ezt a fájlt a NetBeans IDE generálta a forráskódból a futtatáshoz. NetBeans IDE-ből is van lehetőségünk futtatni a programot, és a forráskódot is olvashatjuk. Ezt a következőképp tehetjük meg: NetBeans IDE-ben File menü - Open Project…, betallózzuk a projekt mappát (SudokuSolver mappa). Miután betöltött a Project ablakban megjelenik maga a projekt, és a Source Packages – sudokusolver fül kiterjesztésével böngészhetünk a java kiterjesztésű forrásfájlok között. Futtatáshoz kattintsunk jobb egérgombbal a projekt nevére és válasszuk a Run parancsot.

5.3. Hardver követelmények

A futtatáshoz az operációs rendszer és az aktuálisan futó programok hardverigényén kívül 64MB RAM-ra van szükség.

5.4. A program használata



15. ábra: A program grafikus felülete.

5.4.1. Menük

Fájl menü

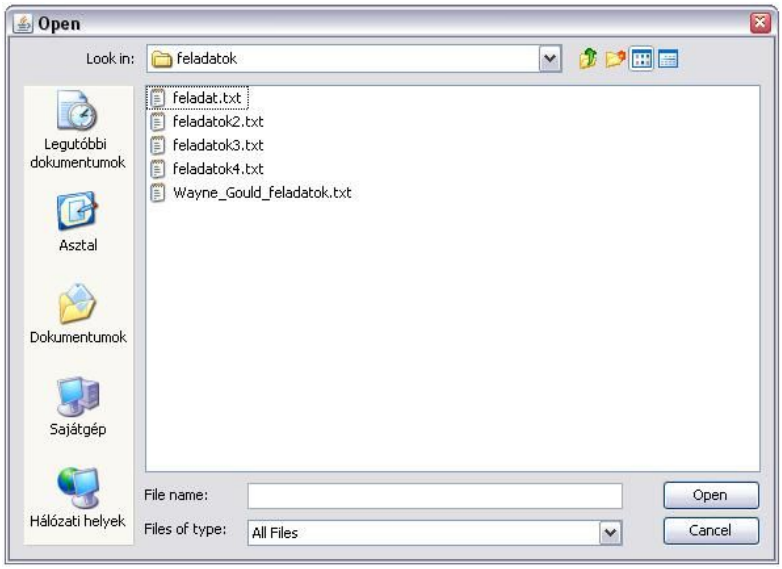
16. ábra: Fájl menü.

Új Tábla

Alapállapotba hozza a program panelt. Kinullázza a táblát, heurisztikák chechkbox-ait kijelöli. Az eddig beolvasott feladatok és megoldásaik is törlésre kerülnek. Ezt akkor használjuk, ha közvetlenül a panelba szeretnénk beírni a Sudoku feladatot, de előtte már egy másik feladvány szerepel a táblában. Közvetlen indítás után a panel alapállapotban van, nem szükséges a parancsot használni.

Fájl Betöltése

Megjelenik a rendszer alapértelmezett fájlválasztója. Kiválasztjuk a feladatokat tartalmazó fájlt. A program betölti a memóriába és előkészíti a fájlban lévő adatot (Sudoku feladatok) a megoldásra.



17 ábra: Fájl megnyitása.

A program input fájlként egy szöveges dokumentumot vár el (txt). A fájlnak a következő módon kell tartalmazni a Sudoku feladatokat: soronként egy feladvány, a 81 érték sorfolytonosan írva, kitöltetlen cella helyén „0” vagy „.” szerepel (17. ábra). A program a beolvasott feladatokat abban a formában kísérli megoldani, ahogy a fájlban szerepelnek, ezért szabályos feladatnak kell lenniük. Szabályos feladatnak tekintem azt a feladatot, amiben a megadott számok a szabályoknak megfelelően helyezkednek el, nincsenek ellentmondások, és egyértelmű megoldása van.

```
.61.3..2..5...81.7.....7.34..9..6.78..32.95..57.3..9..19.7.....8.24...6..4..1.25.
1..83...257...1......5.9.647.4..859...3.1.4...514..3.636.7.4......6...798...52..3
.3...7..46.2.41....5..3.967.4...3..6.87...35.9..7...2.718.2..4....16.8.94..5...3.
.85...21..94.12..3...3..7.45.34.9....4.2.6.3....1.39.76.8..5...1..84.36..27...89.
```

18. ábra: Input fájl tartalma.

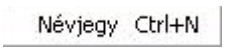
Output megnyitása

Megnyitja a futtatási mappában található alapértelmezett output.txt fájlt Windows jegyzettömbben (notepad). Ennek a menüpontnak a futtatása rendszer specifikus, ezért csak Windows operációs rendszer esetén működik. Az output.txt fájlt manuálisan is megnyithatjuk.

Kilép

A program bezárása.

Súgó menü



19. ábra: Súgó menü.

Névjegy

A program névjegye.

20. ábra: Névjegy.

5.4.2. A főpanel elemei

Tábla (table)

A 9 * 9-es méretű Sudoku tábla. Itt jelenik meg a fájlból beolvasott Sudoku feladat és a megoldása is itt látható. Amennyiben nem fájlból olvassuk be a feladványt manuálisan is beírhatjuk.

Mindent megold gomb (button)

A gomb megnyomásának hatására a fájlból beolvasott összes rejtvényt megpróbálja megoldani a program a kiválasztott heurisztikákkal. A táblában a legutoljára megoldott rejtvény kerül. Az eredmények és a statisztika automatikusan az output.txt fájlba íródnak. Amennyiben nincs beolvasott feladat, az üzenet dobozban ezt jelzi a program.

Kitöltött / Kitöltetlen gomb (togglebutton)

Átállítható gomb. Minden megold gomb alkalmazása után a táblán válthatunk az aktuális feladat nézetén: megjeleníthetjük a kitöltött, vagy kitöltetlen feladványt. Ha nincs beolvasott feladat, vagy még nem került megoldásra, az üzenet dobozban jelzést kapunk róla.

Előző és **Következő** gombok (button)

Fájlból történő beolvasás után átléptet az előző vagy következő feladatra. Előző, vagy következő feladat hiányát jelzi a program.

Heurisztika jelölődobozok (checkbox)

A jelölődobozok két feladatot látnak el. Fájlból történő megoldás után léptetésnél azok a dobozok vannak bejelölve, amelyeket a program használt az adott rejtvény megoldására. A másik felhasználása akkor kerül előtérbe, hogyha nem fájlból olvasunk be, hanem konkrétan beírjuk a táblába a feladatot és azt akarjuk megoldatni a programmal. Ebben az esetben, ha lépésenként oldatjuk meg a feladatot, a jelölő dobozokkal szabályozhatjuk, hogy az egyes lépésekben milyen stratégiák alkalmazásával keressen lehetőségeket.

A következő három gombot akkor használhatjuk, ha egyesével akarjuk megoldani a rejtvényeket, vagy mi magunk gépeljük be a feladványt.

Beolvas gomb (button)

A táblába beírt értékeket beolvassa a program, a későbbiekben ez lesz a rejtvény, amelyet megoldathatunk a Solverrel.

Következő lépés gomb (button)

A tábláról beolvasott feladat lépésenként történő megoldása. A gomb lenyomásával a táblán alkalmazunk a jelölő dobozoknál bejelölt stratégiák közül a legelsőt, ami eredményes. Az alaplépéseket nem lehet kikapcsolni, és ha nincs egy stratégia se bejelölve, akkor csak azzal próbálkozik. Ha a bejelölt stratégiákkal nem volt eredményes a lépés, akkor az adott heurisztikákkal nem oldható meg a feladat. Abban az esetben, ha az összes heurisztika be van jelölve, de gomb hatására még sincs eredményes lépés, akkor a program nem képes megoldani az adott feladatot.

Megold gomb (button)

A program a kiválasztott heurisztikák alkalmazásával megpróbálja megoldani a beolvasott Sudoku feladatot.

Információs ablak (textfield)

Ebben az ablakban jelennek meg a futás közbeni tájékoztató információk és hibák üzenetei.

6. Fejlesztői dokumentáció

A program Java Desktop Application Project alapjára készült. Ebben az esetben a NetBeans IDE generál egy alap Swing Application Framework-ön alapuló vázat.

6.1. Osztályok leírása

6.1.1. SudokuSolverApp osztály

Generált osztály. Tartalmazza a *main* függvényt. Inicializálja a java grafikus felületet (GUI).

6.1.2. SudokuSolverView osztály

Ez az osztály felelős a főpanel grafikus megjelenítésért, és a panelon található irányító gombok által kiváltott feladatok működéséért. Az alkalmazás gerince.

Adattagjai között szerepel a főablakot alkotó elemek és a program logikai működését segítő tagok.

```
public SudokuTable sudokuTable = null;
```
Az aktuális Sudoku tábla, amin a program dolgozik.

```
public Vector <String> puzzles = new Vector <String>();
```
A fájlból beolvasott rejtvények.

```
public Vector <String> solvedPuzzles = new Vector <String>();
```
A megoldott rejtvények string formátumban.

```
public Vector <boolean[]> usedheurs = new Vector <boolean[]>();
```
A megoldott rejtvényekhez tartozó statisztikai adatok.

```
public int counter = -1;
```
Feladat számláló.

```
public int givencell = 0;
```
Manuális beolvasásnál megadott cellák darabszáma.

```
public boolean showsolved = false;
```
Megoldatlan, megoldott rejtvény közötti váltás segédváltozója.

```
public Heuristics h = new Heuristics();
```
Heurisztika osztály példányosítása, ezután használhatjuk a függvényeit.

Függvényei

```
public SudokuSolverView(SingleFrameApplication app)
```
Az osztály konstruktora. A rendszerfeladatokat felügyeli.

```
private void initComponents()
```
A Design nézetben megtervezett panel generált inicializáló kódja.

```
public Integer[][] readTable()
```
Beolvassa az alkalmazás táblájáról az értékeket.

```
public void setTable (String s)
```
A megadott String-ből beállítja az alkalmazás tábláját.

```
public void clearTable()
```
Alapállapotba állítja az alkalmazást. Kinullázza a főpanel táblájának értékeit, bejelöli a kiválasztó dobozokat, és kiüríti a statisztikához szükséges vektorokat.

```
public void readinTable()
```
Az alkalmazás táblájáról beolvasott értékből létrehoz egy SudokuTable objektumot, amin dolgozhat a megoldó program.

```
public void solvenext()
```
Az aktuális SudokuTable feladványon alkalmaz egy heurisztikát.

```
public void solveAll()
```
Megoldja az aktuális feladatot, amennyiben nem tudja megoldani, ezt felismeri.

```
public void fileChoose()
```
Megjeleníti a fájlválasztót. Fájl választása után alapállapotba hozza a programot, majd betölti a fájlban található feladatokat a *puzzles* vektorba

```
public void solveFromFile()
```

Megoldja a *puzzles* vektorban található összes Sudoku feladatot, feltölti a *solvedPuzzles* és *usedheurs* vektorokat és fájlba írja az eredményt és a statisztikát.

```
public String statistic()
```

A három vektor adattag elemeiből statisztikát készít, amelyet formázott String-ként ad vissza.

```
public void previous()
```

Fájlból beolvasott feladványokon léptet az előzőre.

```
public void next()
```

Fájlból beolvasott feladványokon léptet a következőre.

```
public void openOutput()
```

Megnyitja az output.txt fájlt, amibe a statisztikák kerültek.

```
public void toggleSolved()
```

Fájlból beolvasott feladatok esetén váltja a képernyőt a megoldott és megoldatlan rejtvény között.

```
public void showAboutBox()
```

Megjeleníti a Névjegyet.

6.1.3. SudokuSolverAboutBox osztály

A névjegy dialóg ablak kinézetéért és inicializálásáért felelős.

6.1.4. SudokuTable osztály

Sudoku táblát reprezentáló adatszerkezet. Két adattagot tartalmaz. Az egyik egy 9 * 9-es méretű SudokuCell típusú objektumokból felépülő kétdimenziós tömb.

```
private SudokuCell [][] matrix = new SudokuCell[9][9];
```

A másik adattag egy boolean típusokat tartalmazó tömb. Ebben a tömbben statisztikához szükséges adatokat tárolunk arról, hogy mely stratégiákat használtunk a tábla megoldása során és megoldott-e a rejtvény.

```
private boolean usedHeur[] = {false, false, false, false, false, false,
false, false, false};
```

Függvények

```
public SudokuTable(Integer [][] tomb)
```

Az osztály konstruktora. A bemeneti tömb értékeivel példányosít SudokuCell objektumokat, és ezekkel feltölti az osztály tömb adattagját.

```
public boolean[] getUsedHeur()
```

Visszaadja a usedHeur tömböt, a lekért helyen az értékét felhasználhatjuk, módosíthatjuk.

```
public SudokuCell[][] getMatrix()
```

Visszaadja a Sudoku táblát reprezentáló tömböt.

```
public String sudToFileString()
```

Fájlba írás megkönnyítésére String formátumban, visszaadja a tábla celláinak értékét, mögé írva, hogy megoldott, vagy megoldatlan maradt a feladat. Valamint a megoldáshoz felhasznált stratégiák neveit is felsorolja.

```
public boolean isAllSolved()
```

Igaz értékkel tér vissza a függvény, ha a táblában minden mezőnek van értéke, egyébként hamissal.

6.1.5. SudokuCell osztály

Sudoku tábla egy celláját reprezentálja. Adattagjai:

```
private Vector <Integer> candidates = new Vector <Integer>();
```

A cella jelöltjeit tartalmazó vektor.

`private int value;` - A cella értéke.

`private boolean solved = false;` - Ki van-e töltve a cella.

`private int x;` - A cella x koordinátája a SudokuTable táblában.

`private int y;` - A cella y koordinátája.

Függvények

```
public SudokuCell(int value, int x, int y)
```

Az osztály konstruktora. A bemeneti értékekből beállítja az adattagokat. Ha a value, a cella értéke 0, akkor a cella nem megoldott, a jelöltek vektort is feltöltjük 1-től 9-ig számokkal.

```
public Vector<Integer> getCandidates()
```

Visszatér a jelölteket tartalmazó vektorral.

```
public int getValue()
```

Visszaadja a cella értékét.

```
public boolean isSolved()
```

Megmondja, hogy az adott cella meg van-e oldva.

```
public int getX()
```

Visszaadja, hogy a cella a táblában hányadik sorban van.

```
public int getY()
```

Visszaadja, hogy a cella a táblában hányadik oszlopban van.

```
public void setValue(int x)
```

Beállítja a cella értékét.

```
public void setSolved()
```

A cellát megoldottra állítja.

```
public boolean removeCandidate(int x)
```

Eltávolítja x-et a cella jelöltjei közül. Igazzal tér vissza, ha eltávolította, hamissal, ha x nem szerepelt a jelöltek között.

```
public boolean removeCantidate(Vector<Integer>candidates)
```

A bemeneti vektor által tartalmazott jelölt mindegyikét eltávolítja a mező jelöltjei közül.

```
public boolean removeCandidate()
```

Eltávolítja a mezőben található összes jelöltet.

```
public boolean hasSameCandidates(SudokuCell cell)
```
Igazzal tér vissza, ha a paraméterként megadott mező pontosan ugyanazokat a jelölteket tartalmazza, mint az aktuális cella, egyébként hamis.

```
public boolean hasSimilarCandidates(SudokuCell cell)
```
Igaz értékkel tér vissza, ha a megadott cella tartalmaz legalább két ugyanolyan jelöltet, mint az aktuális, egyébként hamis.

```
public boolean equals(Object o)
```
Megállapítja a bemeneti celláról, hogy az megegyezik-e az aktuálissal. Két cella akkor egyezik meg, ha mind x és y értékeik megegyeznek.

6.1.6. Heuristics osztály

Heurisztikák implementált függvényeit tartalmazó osztály. Mindegyik függvény boolean tipusú értékkel tér vissza. Az érték igaz, ha a függvény futása eredményeként beállította egy cella értékét, vagy sikeresen eltávolított legalább egy jelöltet a tábla valamelyik cellájából. Bemeneti paraméterként megkapják a SudokuTable típusú objektumot, ezen dolgoznak a függvények, valamint a képernyőn megjelenített JTable objektumot. Az első függvény az alaplépést és a Naked Single stragtégiát, a többi függvény a nevében szereplő heurisztikát implementálja.

Függvények

```
public boolean setPossibles(SudokuTable sudokuTable,javax.swing.JTable table)
public boolean hiddenSingle(SudokuTable sudokuTable,javax.swing.JTable table)
public boolean nakedPair(SudokuTable sudokuTable, javax.swing.JTable table)
public boolean hiddenPair(SudokuTable sudokuTable, javax.swing.JTable table)
public boolean pointingPair(SudokuTable sudokuTable, javax.swing.JTable
table)
public boolean boxLineReduction(SudokuTable sudokuTable, javax.swing.JTable
table)
public boolean xWing(SudokuTable sudokuTable, javax.swing.JTable table)
public boolean yWing(SudokuTable sudokuTable, javax.swing.JTable table)
```

6.2. Fejlesztési lehetőségek

A program eredeti funkciója Sudoku feladatok nagyszámú megoldása, és erről statisztika felállítása. Ennek érdekében továbbfejlesztésnél mindenképp figyelembe kell venni új stratégiák implementálását, ami lehetővé teheti még bonyolultabb feladványok megoldását. A végső cél annak az elérése lehet, hogy bármilyen érvényes Sudoku feladatot képes legyen megoldani a program heurisztikus algoritmusokkal.

Az algoritmusok egyszerűen átalakíthatók úgy, hogy 9 * 9-es méretű feladványon kívül más méretűeken is használhatók legyenek.

A felhasználói felület kiegészítésével élvezetes játékká lehet alakítani a programot, ami képes segítséget adni a lépésekben.

A program a heurisztikák és statisztikák használatával megoldás létrehozása közben felbecsülheti a feladat nehézségi fokát, így osztályozhatja a feladatokat. Ez fontos feladat, mert a generáló programok által készült feladatok nehézségi fokát nehéz felmérni a generáló algoritmus által.

A forráskód írása közben törekedtem arra, hogy az algoritmusok a lehető legkevesebb lépést hajtsák végre keresés közben, de további optimalizálással csökkenthető a felesleges lépések száma.

7. Tesztadatok ismertetése

7.1. Források

A programot nyomtatásban megjelent és internetes forrásból származó Sudoku feladványokkal teszteltem. A feladatok osztályozva vannak a készítő által. A tesztek alatt arra voltam kíváncsi, hogy a különböző nehézségű rejtvények megoldásához milyen heurisztikákra van szükség, valamint, hogy egyáltalán képes-e a programom megoldani ezeket a feladatokat.

Teszt adat forrásai:

1. Wayne Gould – Sudoku számrejtvény 1.
2. Wayne Gould – Sudoku számrejtvény 2.

   Wayne Gould híres Sudoku rejtvénykészítő. Miután Japánban megismerkedett a játékkal maga is nekikezdett rejtvényeket készíteni. Fontos lépéseket tett annak érdekében, hogy a Sudoku világszerte ismert és közkedvelt legyen. A két könyvben összesen kétszáz darab feladat található, ami nehézség szerint a következő csoportokba van sorolva: könnyű (8db), közepes (52db), nehéz (90db) és nagyon nehéz (50db).
3. Szudoku Kezdőknek

   A könyv százhuszonöt darab rejtvényt tartalmaz, a következő nehézségi csoportokba sorolva: könnyű (45db), közepes (55db), nehéz (25db). Megjegyzés: A könyv két feladatban nyomtatási, vagy gépelési hibát tartalmaz (a feladatban megadott kezdő számok nem a megadott helyükön szerepelnek a mellékelt megoldó kulcsban). Egyik esetben az egyetlen, valószínűleg elgépelt cella értékének kijavítása után a feladat érvényessé vált, így a javított feladat szerepel a tesztadatok között. A másik feladatban több hiba is van ezért azt a feladatot eltávolítottam a tesztadatok közül.
4. Internetes rejtvénygyűjtemény. Forrás: http:/www.sudoku-puzzles.net/. A feladatok bármilyen célra felhasználhatóak. A gyűjtemény száz darab rejtvényt tartalmaz csoportokra bontva: nagyon könnyű (20db), könnyű (20db), közepes (20db), nehéz (20db), nagyon nehéz (20db).
5. Általam gyűjtött feladatok. Ez a csoport speciálisan válogatott, érdekes feladatokat tartalmaz:

- Van közöttük olyan feladat, aminek megoldása brute-force algoritmussal bizonyítottan közel maximális számú visszalépést igényel, heurisztikus megoldásához azonban a legegyszerűbb stratégiák is elégségesek.
- Tartalmaz néhányat a legnehezebbnek vélt Sudoku rejtvényekből. Ezeket a rejtvényeket találgatáson alapuló stratégiák segítsége nélkül az általam próbált online Sudoku megoldó szoftverek sem voltak képesek megoldani. Egyes feladatokhoz a megadott cellákon kívül képtelenek voltak akárcsak egyet is megfejteni.
- Tartalmazza a példaként használt rejtvényeket is, valamint a fejlesztés alatt tesztelésre használt feladatokat.

7.2. Statisztikák

A program a statisztikai adatokat az output.txt fájlban tárolja (19. ábra). A fájl tartalmazza a megoldott feladatot (a bementhez hasonlóan), valamint hogy milyen stratégiákat alkalmazott a program a megoldása során. A feladatcsoportok végén összesítő statisztikát olvashatunk.

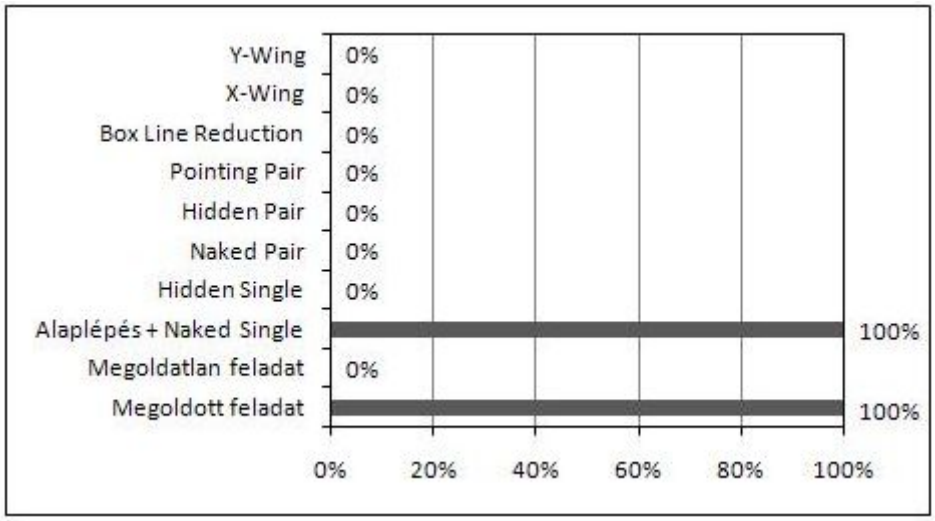
```
################################################################################################
2009.11.24 16:58:39
235971486461382597987546231796254318518693742342718965629135874174829653853467129:megoldva;
Hasznalt strategiak: Alaplépések Naked Single Hidden Single Naked Pair Hidden Pair Pointing Pair Box Line Reduction
################################################################################################
Aktuális Statisztika: Bemenet: 1 db rejtvény
Megoldott: 1 (100%); Megoldatlan: 0 (0%)
Stratégia használat:
Alaplépés + Naked Single: 1 (100%)
Hidden Single: 1 (100%)
Naked Pair: 1 (100%)
Hidden Pair: 1 (100%)
Pointing Pair: 1 (100%)
Box Line Reduction: 1 (100%)
X-Wing: 0 (0%)
Y-Wing: 0 (0%)
################################################################################################
```

21. ábra: Output.txt tartalma.

Tesztelés folyamán statisztikát készítettem az egyes nehézségi szintekbe eső feladatok megoldásához szükséges heurisztikákról. A diagramokon az y tengelyen olvasható megoldott, valamint megoldatlan feladatok százalékos aránya, valamint a heurisztikák alulról felfelé, a legegyszerűbbtől a bonyolultabbakig. Az összes heurisztika használata esetén a használat sorrendje is így alakul. Az x tengely azt mutatja, hogy a megoldáshoz, a Sudoku feladványok hány százalékánál volt szükség az adott stratégiára.

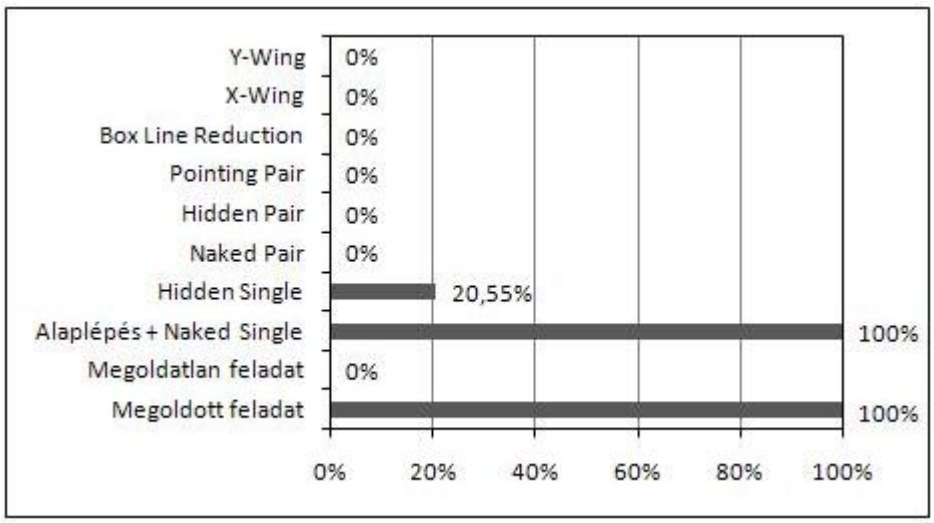
Nagyon könnyű feladatok

Ebben a csoportban húsz darab feladat van. A diagramon látható, hogy a feladatok megoldásához elegendő volt az alaplépés és a legegyszerűbb heuriszitka, a Naked Single. Elegendő tisztában lennünk a Sudoku szabályaival, bármilyen egyéb tudás használata nélkül képesek leszünk megoldani egy ilyen nehézségű feladatot. Tökéletes nehézség a játékkal újonnan ismerkedők számára.



22. ábra: Heurisztikák használata nagyon könnyű feladatok esetén.

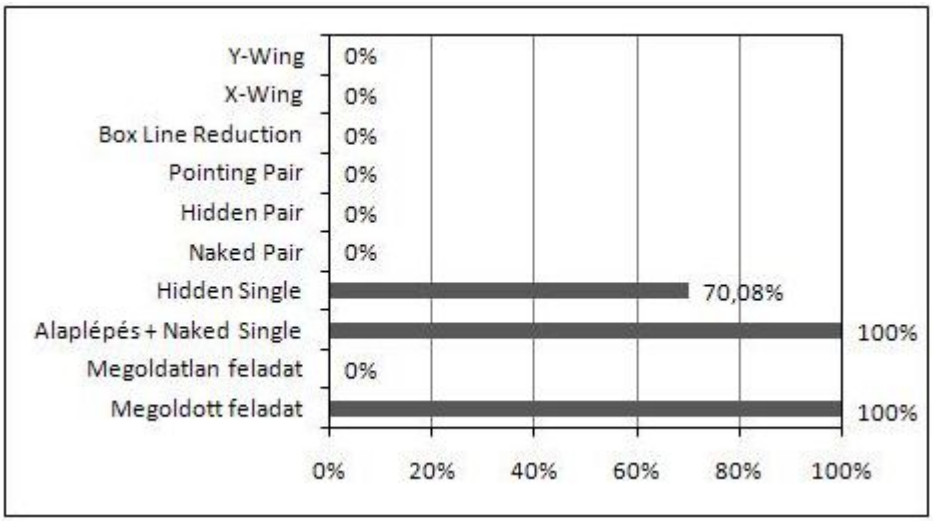
Könnyű feladatok



23. ábra: Heurisztikák használata könnyű feladatok esetén.

A hetvenhárom feladat megoldásához a feladatok egy ötödénél szükség van Hidden Single stratégia alkalmazására.

Közepes feladatok



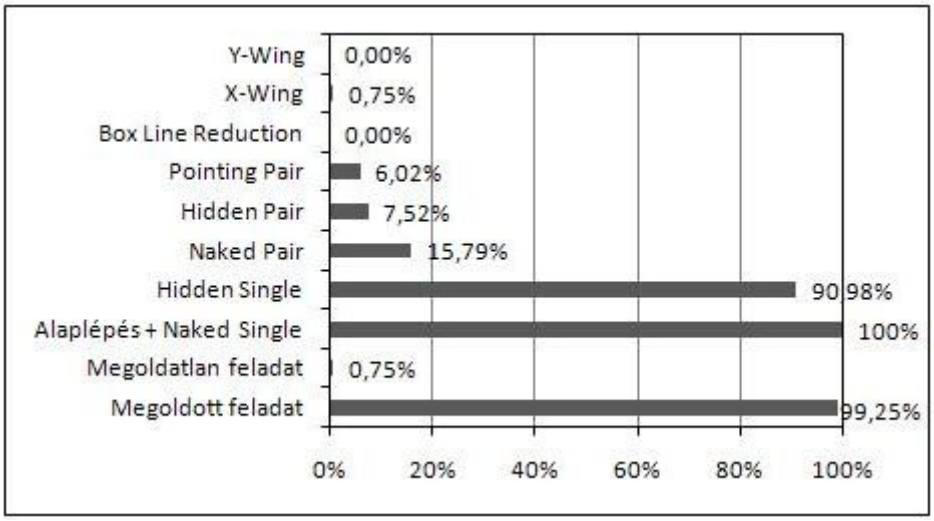
24. ábra: Heurisztikák használata közepes feladatok esetén.

Érdekes megfigyelni, az alaplépés és Naked Single stratégiákon kívül a közepes feladatok (127db) megoldásához is elegendő a Hidden Single heurisztikát használni. A rejtvények több mint kétharmadánál volt szükség rá. A közepes feladatok megoldása során, ahol szükség volt a plusz stratégia alkalmazására átlagosan többször is kellett használnia, mint könnyű feladatok estén.

Megfigyelhető tehát, hogy egészen a közepes nehézségű feladatok megoldásáig elegendő volt a Naked Single és Hidden Single heurisztikák alkalmazása. Amennyiben egy játékos csak ezt a két stratégiát ismeri még akkor is képes az ilyen szintű feladatok megoldására. Itt megjegyezném, hogy bár a Hidden Single stratégia meglehetősen egyszerűnek tűnik, és a számítógépnek a memóriája miatt az is, két Hidden Single felismerése között megoldás közben egy ember számára nehézségbeli különbségek vannak. Ha például a vizsgált házban csak két üres cella van, akkor csak két értéket, és a vizsgált ház és a cellákat tartalmazó többi házat kell átvizsgálni. Ezt könnyedén képes követni egy játékos jegyzetelés nélkül, azonban ha több az ismeretlen cella, megnövekedik a vizsgálandó cellák száma, amit nehezebb fejben tartani.

Nehéz feladatok

A százharminchárom darab nehéz feladat már igazi próba elé állította a megoldó programot. A rejtvények között akadt egy, amelyet a program nem volt képes megoldani, és a több heurisztika alkalmazása is szükségessé vált. Tovább nőtt a Hidden Single felhasználási aránya, elérte a 90%-ot. Amennyiben kikapcsoljuk a Hidden Single-től bonyolultabb stratégiákat a feladatok 18%-a megoldatlan marad. Ebből látható, hogy egyes feladatoknál több bonyolult stratégiát is alkalmazott a program.

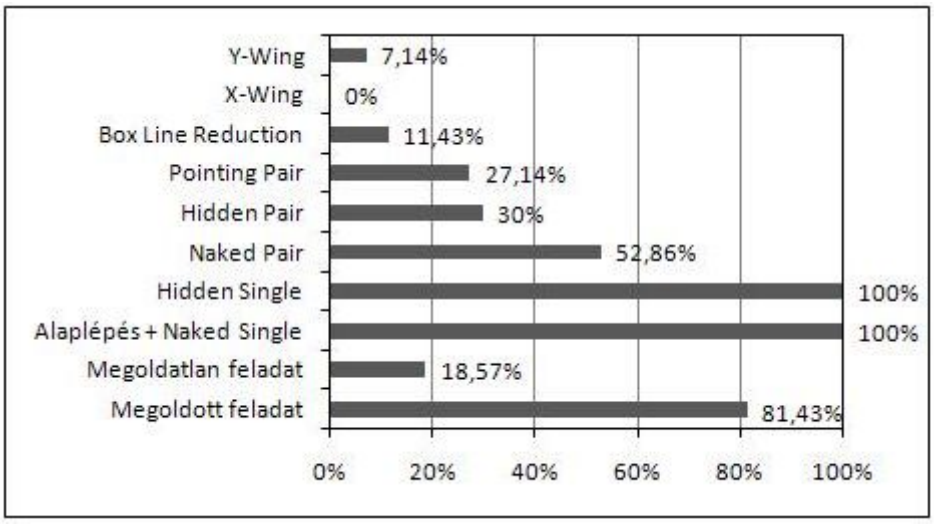


25. ábra: Heurisztikák használata nehéz feladatok esetén.

Látható, hogy ha felkeltették az érdeklődésünket a nehéz feladatok, már szükség van arra, hogy megismerkedjünk további három heurisztikával. Ezen öt heurisztika ismeretében már bátran neki kezdhetünk nehéz feladatoknak.

Nagyon nehéz feladatok

Hetven darab nagyon nehéz feladatot tartalmaz a teszteset. Látható, hogy a bonyolultabb heurisztikák használata igen csak megszaporodott, de ennek ellenére majdnem a feladatok 19%-a megoldatlanul maradt. A megoldatlan feladatokat ellenőriztem interneten található évek alatt fejlesztett Sudoku megoldó rendszerekkel. A rejtvények megoldhatók heurisztikus módszerekkel, de a korábban ismertetetteken kívül még számos igen bonyolult algoritmusra volt szükség.

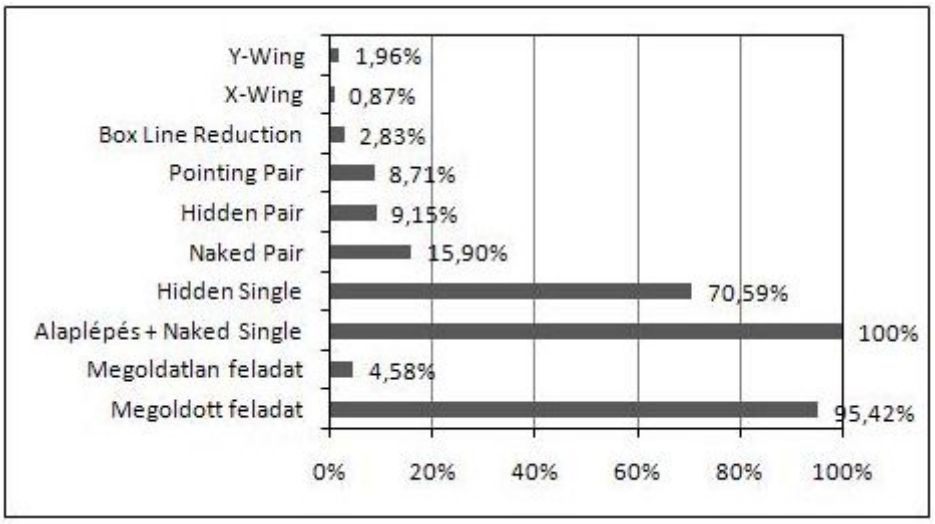


26. ábra: Heurisztikák használata nagyon nehéz feladatok esetén.

Az utolsó feladatok nehézségének már nincsen felső határa. Az összes általam ismertetett heurisztika ismerete sem biztos, hogy elegendő egy nagyon nehéz rejtvény megoldásához. Jól látszik, hogy a legbonyolultabb stratégiák (X-Wing, Y-Wing) használata elég csekély számú. A megoldatlan feladatok elemzése közben megfigyeltem, hogy a megoldáshoz szükséges heurisztika halmazaik metszete páronként maximum egy – két módszert tartalmaz, és mindig van a metszeten kívül álló stratégia. Ebből következik, hogy gondosan, egyedien vannak megalkotva a feladatok, mindig lesz új kihívás.

Összesített statisztika

A következő statisztikában szerepel az összes Sudoku rejtvény (459db), amivel próbára tettem a programot. Az eddig felsorolt feladatok és azok a különleges feladatok is, amiket különböző forrásokból egyesével gyűjtöttem. Ezek nem szerepeltek a korábbiak között, mert a forrásban nincsenek osztályba sorolva.



27. ábra: Heurisztikák használata minden feladattípus esetén.

Érdekes megfigyelés

Alaplépést és Naked Single-t használva, a többi heurisztikát kikapcsolva, a rejtvények 28,98%-a oldható meg. Bekapcsolva még a Hidden Single stratégiát a feladatok 78,21%-a lesz megoldható. Amennyiben a Hidden Single-ön kívül minden más módszert alkalmazunk a feladatok 81,7%-át oldjuk meg. Ebből látható, hogy a Hidden Single egy nagyon erős stratégia.

8. Összefoglalás

A Sudoku egy világszerte népszerű számrejtvény. Egy 9 * 9-es méretű táblából áll, ami kilenc darab 3 * 3-as méretű blokkra oszlik. A tábla néhány cellájában adva van egy szám 1-től 9-ig. A feladat teljesen kitölteni a táblát úgy, hogy minden sorban, oszlopban és blokkban a számok 1-től 9-ig pontosan egyszer szerepeljenek.

Diplomadolgozatom elkészítésével betekintést szerettem volna nyújtani a Sudoku rejtvények megoldásához szükséges heurisztikák világába. A heurisztikák az emberi logikát utánozva működnek. Egy program által heurisztikákkal megoldott rejtvényt, egy logikusan gondolkodó ember is képes megoldani.

Bemutattam számos stratégiát (Naked Singe, Hidden Single, Naked Pair, Hidden Pair, Pointing Pair, Box Line Reduction, X-Wing, Y-Wing). Néhányról kiderült, hogy ismeretük elengedhetetlen a rejtvények megoldásához, ilyen a Naked Single mellett a Hidden Single.

Készítettem egy Sudoku rejtvényt megoldó programot. A célom az volt, hogy ez a program képes legyen tisztán heurisztikus algoritmusokkal megoldani bármilyen nehézségű rejtvényt. A programmal szemben támasztott fő követelmény, hogy alkalmas legyen nagy mennyiségű rejtvény megoldásán túl az alkalmazott heurisztikákról statisztikák készítésére. A program a bemenő rejtvények 95%-át volt képes megoldani. A megoldatlan feladatok megoldásához további heurisztikák ismertetésére és implementálására lett volna szükség.

A statisztikák jó képet adtak arról, hogy az egyes nehézségi csoportba tartozó rejtvények megoldásához mely heurisztikák ismerete elengedhetetlen. Egészen közepes nehézségig elegendő ismerni a Naked Single és Hidden Single heurisztikákat. A nehéz feladatok megoldásához szükség van még a Naked Pair, Hidden Pair, Pointing Pair és Box Line Reduction stratégiák használatára. Ahhoz, hogy boldoguljunk a legnehezebb feladatokkal már elengedhetetlen a legbonyolultabb heurisztikák alkalmazása (X-Wing, Y-Wing és még további diplomadolgozatomban nem ismertetett stratégiák).

9. Irodalomjegyzék

[1] http://hu.wikipedia.org/wiki/Sudoku – Wikipedia, Letöltve 2008. szeptember

[2] http://en.wikipedia.org/wiki/Sudoku – Wikipedia, Letöltve 2008. szeptember

[3] http://hu.wikipedia.org/wiki/Latin_négyzet – Wikipedia, Letöltve 2009. szeptember

[4] http://en.wikipedia.org/wiki/Algorithmics_of_sudoku – Wikipedia, Letöltve 2008. szeptember

[5] http://hu.wikipedia.org/wiki/Heurisztika – Wikipedia, Letöltve 2009. szeptember

[6] http://www.ecclestoad.co.uk/blog/2005/06/02/sudoku_solver_in_three_lines_explained.html – Sudoku solver in three lines explained – 2005 Eccles & Toad, Letöltve 2008. szeptember

[7] www.math.u-szeged.hu/Sudoku/sudoku.pdf – Makay Géza, SZTE, Bolyai Intézet

[8] http://en.wikipedia.org/wiki/Wayne_Gould – Wikipedia, Letöltve 2009. november

[9] http://www.sudokuwiki.org/ - Andrew Stuart, Scanraid Ltd, 2009, Letöltve 2009. november

[10] http://toys.brando.com/prod_detail.php?prod_id=00112 – 3D Sudoku Cube, 2009 Brando Workshop

Heurisztikák leírása

[11] http://www.sudokuwiki.org/sudoku.htm – Sudoku Solver by Andrew Stuart, Scanraid Ltd 2005 - 2009

[12] http://www.sudokuwiki.org/Strategy_Families – Strategy Families, Andrew Stuart, Scanraid Ltd, 2008. április, Letöltve 2008. szeptember

[13] http://www.sudokuwiki.org/Naked_Candidates – Naked Candidates, Andrew Stuart, Scanraid Ltd, 2005. június, Letöltve 2008. szeptember

[14] http://www.sudokuwiki.org/Hidden_Candidates – Hidden Candidates, Andrew Stuart, Scanraid Ltd, 2008. április, Letöltve 2008. szeptember

[15] http://www.sudokuwiki.org/Intersection_Removal – Intersection Removal, Andrew Stuart, Scanraid Ltd, 2008. április, Letöltve 2008. szeptember

[16] http://www.sudokuwiki.org/X_Wing_Strategy – X-Wing Strategy, Andrew Stuart, Scanraid Ltd, 2008. április, Letöltve 2008. szeptember

[17] http://www.sudokuwiki.org/Y_Wing_Strategy – Y-Wing Strategy, Andrew Stuart, Scanraid Ltd, 2008. április, Letöltve 2008. szeptember

[18] http://www.sudokuwiki.org/Multivalue_X_Wing_Strategy – Multivalue X-Wing Strategy, Andrew Stuart, Scanraid Ltd, 2008. április, Letöltve 2008. szeptember

[19] http://www.sudokuwiki.org/Y_Wing_Chains – Y-Wing Chains, Andrew Stuart, Scanraid Ltd, 2008. április, Letöltve 2009. október

[20] http://theory.tifr.res.in/~sgupta/sudoku/algo.html – Sudoku Tips: How to solve Sudoku, The Mathematics of Su Doku, Sourendu Gupta, 2005, Letöltve 2008. szeptember

[21] http://www.palmsudoku.com/pages/techniques-11.php – Solving Sudoku: Technique 11: Nishio – Astraware Limited, 2008, Letöltve 2009. október

Tesztadatok forrásai

[22] Wayne Gould: Sudoku Számrejtvény 1. Partvonal Könyvkiadó, Budapest, 2005, ISBN 963 868 796 7

[23] Wayne Gould: Sudoku Számrejtvény 2. Partvonal Könyvkiadó, Budapest, 2005, ISBN 963 868 797 5

[24] Szudoku Kezdőknek, Pécsi Direkt kft. Alexandra Kiadója, 2005, ISBN 963 369 466 3

[25] http:/www.sudoku-puzzles.net/ - Sudoku, Kakuro & Futoshiki Puzzles, M. Flueckiger, 2008, Letöltve 2009. október

[26] http://www.websudoku.com/ - Web Sudoku – Billions of Free Sudoku Puzzles to Play Online, 2005